CSE474

Project Report

Name: Md. Farhadul Islam

ID: 19201086

Representing Uncertainty in Deep Learning Models using Monte Carlo Dropout

Md. Farhadul Islam[0000−0003−3249−4490]

School of Data and Sciences, BRAC University
md.farhadul.islam@g.bracu.ac.bd

Abstract. Dropout is a technique for preventing overfitting in a deep learning model. At each update of the training phase, dropout functions by setting the outgoing neurons that make up hidden layers to 0. In the field of applied machine learning, deep learning technologies have gotten a lot of attention. However, such regression and classification algorithms do not account for model uncertainty. Bayesian models, on the other hand, provide a theoretically based framework for reasoning about model uncertainty, but they generally come at a high computing expense. The Monte Carlo (MC) dropout technique provides a scalable way to learn a predictive distribution. In MC dropout, the dropout is applied at both training and test time unlike the normal dropout. The prediction is no longer deterministic at test time, but rather depends on the nodes you pick at random. As a result, the model may predict different values each time given the same datapoint. In this experiment, we create a simulation to show the uncertainty of a certain model considering different conditions. We run our experiment on the California Housing Dataset to represent the uncertainty of a model under different conditions.

Keywords: Deep Learning · Neural Network · Monte Carlo Dropout

1 Introduction

In statistics, the aphorism "all models are wrong" is typically extended to "all models are wrong, but some are useful".This aphorism is a famous quote often attributed to the British statistician George E. P. Box. The aphorism acknowledges that statistical or scientific models will always fall short of reality's intricacies, yet they can nevertheless be useful. In deep learning, our predicted outputs are not deterministic. The stochastic nature of deep learning makes the model uncertain. Knowing the uncertainty of a model is very crucial when it comes to decision making. When dealing with medical, financial data the uncertainty can cause serious complications if the uncertainty of a deep learning model is not represented.

Deep learning models usually lack the representation of uncertainty and make overconfident and faulty predictions [3,4].

Bayesian techniques like Bayesian neural networks (BNNs) and Gaussian processes have gotten a lot of press for providing uncertainty measurements in addition to predicting probabilities. BNNs and Gaussian processes, unlike single predictions, produce predictive distributions in which BNN weights are combined with priors distribution [2], whereas Gaussian processes include priors over functions [5]. These methods are effective theoretically, but tough to do in practice.

Gal and Ghahramani [1] have proposed a very simple and easy to implement method for representing model uncertainty. They have introduced MC dropout, which is basically normal dropouts used in the testing phase of the model. Our experiment use this method to represent uncertainty in deep learning models.

Our experiment focuses on representing uncertainty when the features, hyperparameters and size of the model is changed. We build a framework to simulate the representation, where we consider different conditions like, using single feature or multiple feature, using lightweight or heavyweight architecture, changing the hyperparameters of a certain model. Our study shows both prediction and uncertainty representation of a regression task. This allows us to take safer decisions than before because we get to know how confident the deep learming model is.

2 Literature Review

There are several studies on representing uncertainty in deep learning models. Such as Bayesian Methods, SGD Based Approximations, Methods for Calibration of DNNs.

2.1 Bayesian Methods

In Bayesian model averaging, a distribution is placed over model parameters, and then these parameters are marginalized to build a full predictive distribution. The foundational studies of Neal [2] and MacKay [6] established Bayesian approaches as the state-of-the-art approach to learning using neural networks in the late 1990s. Modern neural networks, on the other hand, frequently include millions of parameters, and the posterior over these parameters (and hence the loss surface) is very non-convex, necessitating mini-batch techniques to get to a suitable solution space [7]. Bayesian techniques have been essentially intractable for current neural networks as a result of these factors.

Markov chain Monte Carlo (MCMC): MCMC was very popular for inference with neural networks, through the Hamiltonian Monte Carlo (HMC) work of Neal [2]. However, HMC requires full gradients, which is computationally intractable for modern neural networks. Chen et al. [8] presented stochastic gradient HMC (SGHMC), which allows stochastic gradients to be employed in Bayesian inference, which is important for both scalability and exploring a space of solutions with high generalization. In the stochastic gradient situation,

stochastic gradient Langevin dynamics (SGLD) [9] employs first order Langevin dynamics. In the case of indefinitely tiny step sizes, both SGHMC and SGLD asymptotically sample from the posterior. Using limited learning rates causes approximation problems in reality, and tweaking stochastic gradient MCMC algorithms can be tricky.

Variational Inference: Graves [5] proposed that the weights of neural networks be fitted with a Gaussian variational posterior approximation. The reparameterization strategy was suggested by Kingma and Welling [10] for training deep latent variable models, and numerous variational inference techniques based on the reparameterization trick were developed for DNNs. While variational approaches function well on modestly sized networks, they are challenging to train on bigger designs such as deep residual networks, according to other studies.

Dropout Variational Inference: Gal et al. [1] propose a novel theoretical framework that casts deep neural network (NN) dropout training as approximate Bayesian inference in deep Gaussian processes. As a direct outcome of this theory, we may use dropout NNs to represent uncertainty, retrieving information from current models that has previously been discarded. This solves the challenge of conveying uncertainty in deep learning without losing test accuracy or computational complexity. This approach is very easy to implement and used widely.

SWA-Gaussian for Bayesian Deep Learning: Maddox et al. [12] propose SWA-Gaussian (SWAG) for Bayesian model averaging and uncertainty. Stochastic Weight Averaging (SWA), which iterates with a modified learning rate schedule and computes the first moment of stochastic gradient descent (SGD), has recently been proven to increase generalization in deep learning. With SWAG they form an approximate posterior distribution over neural network weights by fitting a Gaussian using the SWA solution as the first moment and a low rank plus diagonal covariance also derived from the SGD iterates. Then, they sample from this Gaussian distribution to perform Bayesian model averaging. In line with studies defining the stationary distribution of SGD iterates, their experiments show that SWAG approximates the form of the real posterior estimation.

2.2 SGD Based Approximations

Mandt et al. [13] proposed to use the iterates of averaged SGD as an MCMC sampler, after analyzing the dynamics of SGD using tools from stochastic calculus. Chen et al. [14] investigate the problem of statistical inference of true model parameters based on SGD when the population loss function is strongly convex and satisfies certain smoothness conditions.

2.3 Methods for Calibration of DNNs

For improved calibration, Lakshminarayanan et al. [15] proposed utilizing ensembles of many networks, as well as an adversarial loss function to be utilized

when possible. Outside of probabilistic neural networks, Guo et al. [3] developed temperature scaling, a process that rescales the logits of DNN outputs for improved calibration using a validation set and a single hyperparameter. Using a similar rescaling method, Kuleshov et al. [16] offer calibrated regression. These methods are quite tougher to implement comparatively.

3 Methodology

Our proposed methodology is a step by step representation of a model's uncertainty. We have to use the regular Dropout and keep it on during the test phase to represent the uncertainty. For an in depth analytical experiment we create two different neural network models of different sizes. We will use different features to see the impact of using them.

Layer	Output Shape	Parameters
Dense	(None, 100)	200
Dropout / MC Dropout	(None, 100)	0
Dense	(None, 200)	20200
Dropout / MC Dropout	(None, 200)	0
Dense	(None, 200)	40200
Dropout / MC Dropout	(None, 200)	0
Dense	(None, 100)	20100
Dropout / MC Dropout	(None, 100)	0
Dense	(None, 1)	101
Total		80,801

Table 1: Model Summary of Lightweight Regular & MC Model

3.1 Monte Carlo Dropout

Gal et al. [1] interpret dropout as a sampling method that is equivalent to a variational approximation of a deep Gaussian process. A deep Gaussian process is a Bayesian machine learning model that ordinarily outputs a probability distribution, and conventional dropout at test time may be used to determine properties of this underlying distribution. The estimated variance of the distribution is used to determine the model's uncertainty for a given input. Monte Carlo dropout is the name for this approach of estimating uncertainty. A neural network is initially trained regularly using conventional dropout before being used to implement Monte Carlo dropout. The network is run T times with conventional dropout, all with the same input but different randomly generated dropout masks each time, to perform inference on an input sample. In simple words, Monte Carlo dropout keeps the functionality of a regular dropout turned on during the test phase as well. It helps to generate random predictions and

Layer	Output Shape	Parameters
Dense	(None, 1000)	2000
Dropout / MC Dropout	(None, 1000)	0
Dense	(None, 1000)	1001000
Dropout / MC Dropout	(None, 1000)	0
Dense	(None, 2000)	2002000
Dropout / MC Dropout	(None, 2000)	0
Dense	(None, 2000)	4002000
Dropout / MC Dropout	(None, 2000)	0
Dense	(None, 1000)	201000
Dropout / MC Dropout	(None, 1000)	0
Dense	(None, 1)	1001
Total		9,009,001

Table 2: Model Summary of Heavyweight Regular & MC Model

interpret them as samples from a probabilistic distribution. This is a Bayesian process.

3.2 Artificial Neural Network

We create two separate neural network models to show the difference of the generated graphs of uncertainty. We create a lightweight and a heavyweight model for both regular and Monte Carlo dropout model. Both regular and Monte Carlo model are the same except for the dropout layer.

We develop our lightweight model with an input layer and we add a Dense layer and a dropout function. We add a dropout after each Dense layer which we will refer as a block. We add total 4 blocks in total, where the units (size of the output) are 100, 200, 200, 100 respectively. Finally, we add a dense layer with a unit 1, because this is the final output layer of the model. We use ReLu as our activation function in each dense layer except the last one. The total number of paramters in this model is 80,801.

For the heavyweight model, we add an extra block with more units. We add total 5 blocks, where the units are 1000, 1000, 2000, 2000, 1000 respectively. Similarly, we add a dense layer with a unit 1, because this is the final output layer of the model. We use ReLu as our activation function in each dense layer except the last one in the heavyweight model as well. The total number of paramters in this model is 9,009,001, which is very much larger than the lightweight one we mentioned.

We use Early Stopping to avoid training our model for too long. Early stopping is a method that specifies an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset.

3.3 Representing Uncertainty

We use three single features at first and compare their graphs, generated from the two different models. Then, we use one, multiple and all features in a heavyweight model to show how the number of features can make an impact in predictions. The whole workflow is shown in Figure 1.

4 Experiment and Results

4.1 Experimental Setup

The training and testing phases for this NN model are developed using the Python[20] packages including Tensorflow[17]/Keras[18]. The data preprocessing part of the experiment was conducted through Scikit-learn [19]. The models are trained and tested on an NVIDIA GeForce RTX 3080 Ti with a performance of 34.1 TeraFLOPs.

4.2 Dataset

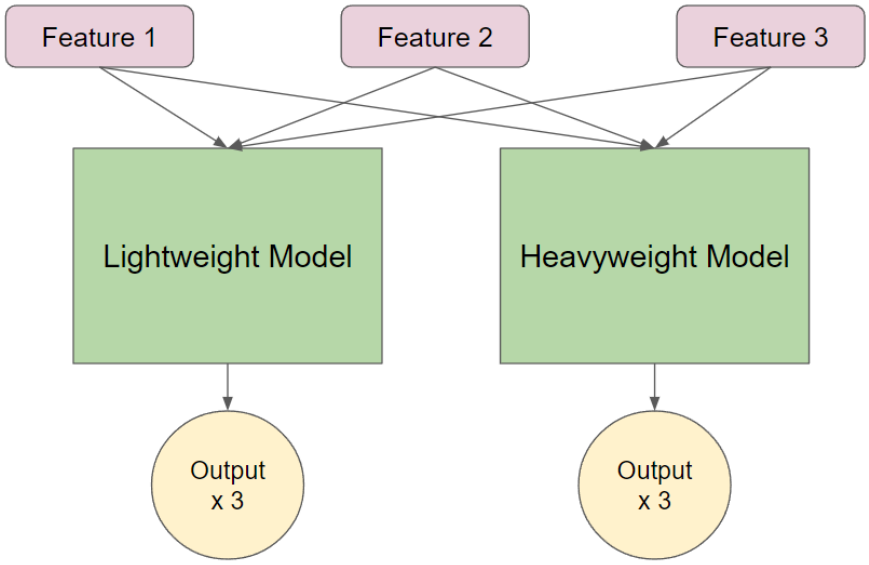
Dataset used in this experiment is known as 'California Housing Dataset'. This dataset was initially featured in [27]. This dataset has 9 feature columns, where the 'medianHouseValue' is column generally taken as the target. In our experiment, we use a split dataset with a train to test ratio of 7.5:1.5. There are 17000 train data and 3000 test data used in the experiment.

4.3 Result Analysis

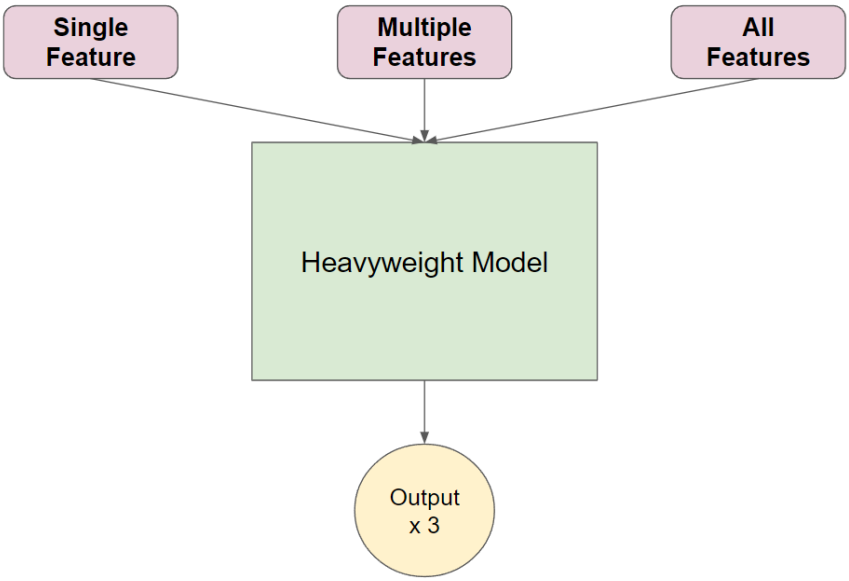
As the goal and motivation of this experiment is to represent uncertainty, we do not focus on the evaluation metrics such as Accuracy, Mean Squared Error etc. The models were created to show the difference of uncertainty between two different models and not to achieve high performance. In Table 3, we show the epochs taken to train each model in different scenarios.

In Figure 2.(a), we select the feature "Population". We get two separate results here. Taking decisions from the regular prediction is risky and tough, as we can no be sure which one is the better fit. But when we look at our MC model representations, we can be sure that the heavyweight model is the better model as the model is less spread and more packed comparing to the lightweight MC model. In Figure 2.(b), we select the 'Median Income" feature. In this case the predictions look pretty similar. But the uncertainty graphs gives us a more concrete visualization of the strength of these two models. But in figure 2.(c), the decisions are pretty similar until we increase the input value. In this one, the model gets very uncertain when the input value gets high. This is a very helpful representation for us, as we can not get these information using regular neural networks.

Now, in Figure 3, we use different number of features to train our model and



(a) We take 3 single features separately in both models and generate graphs for each



(b) We take single, multiple, all features in consideration and generate graphs for each

Fig. 1: Methodology Workflow

the results we get are quite self explanatory here. In 3.(a) the model is quite confident with one feature (which is avery important feature for this dataset). The distribution looks more packed than the other two. In 3.(b) the distribution is pretty similar but more wide and spread. When we use all features to train our model, the model gets very uncertain as there are unimportant features in the dataset and multiple features can add unwanted factors to the model. In 3.(c) we get to see the result of taking all features.

The results lead us to conclude that representing uncertainty considering different conditions can help us to take major decisions with less risks.

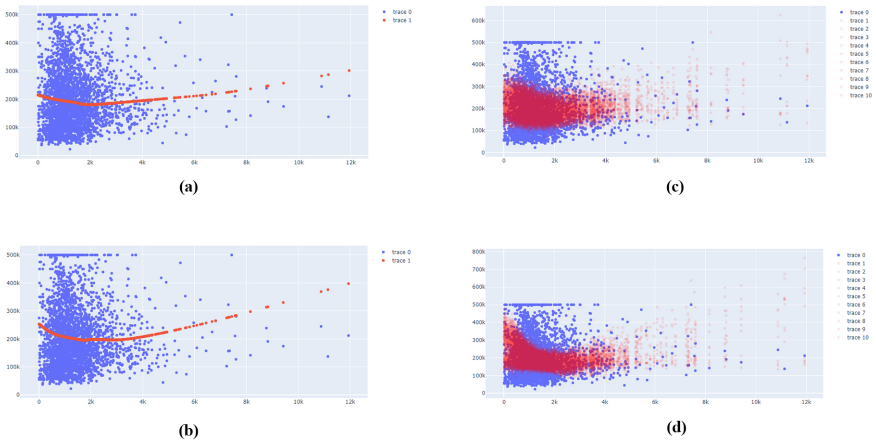
Feature Selected	Lightweight Model	Heavyweight Model	Lightweight MC Model	Heavyweight MC Model
Population	23	12	25	11
Median Income	23	12	22	13
Total Rooms	16	23	32	21
Multiple	39	31	31	14
All	22	19	28	15

Table 3: Number of epochs for different tests

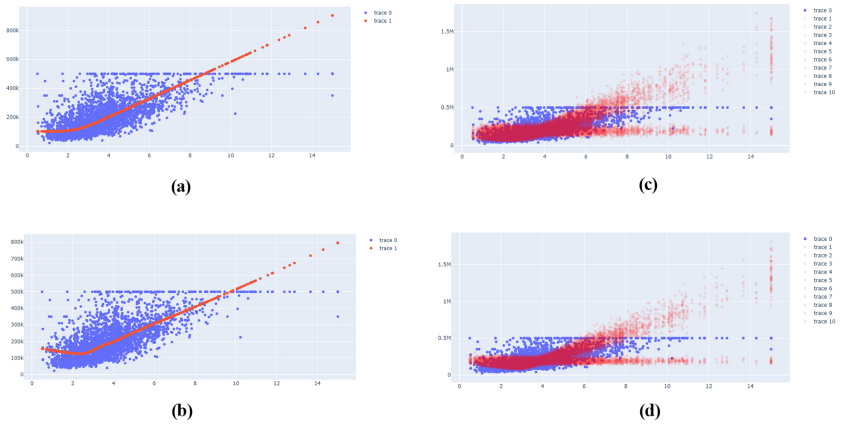
5 Applications and Further Improvements

MC dropout can be used in many areas of deep learning. Many applications of Monte Carlo dropout have been identified in practice, including time-series prediction [22] and medical imaging [23]. Bayesian neural networks [21] and ensemble-based techniques [15] are two further ways suggested to quantify model uncertainty. Miok et al. [24] proposed a method of generating data with MC dropout. The idea is to incorporate Monte Carlo Dropout method within Autoencoder (MCD-AE) and Variational Autoencoder (MCD-VAE) as efficient generators of synthetic data sets.

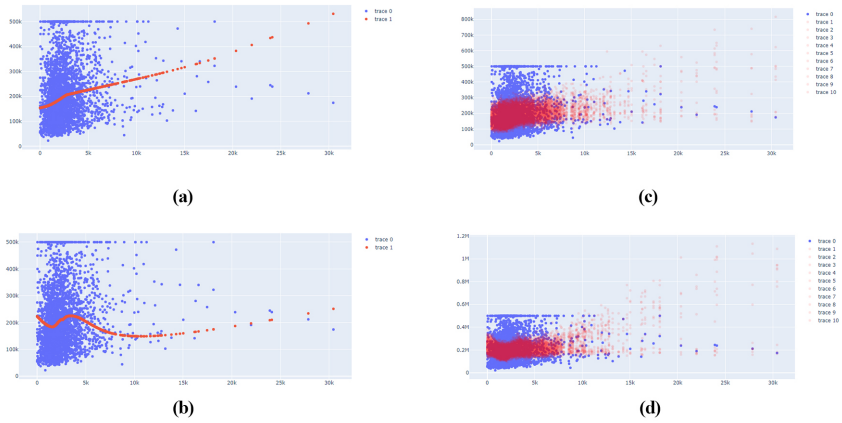
MC dropout is very easy to implement but it has some flaws. Osband et al. [25] discusses the risks regarding MC-dropout. Risk is described as a model's intrinsic variability, whereas uncertainty seems to encapsulate our unsure conviction in the forecast value. Consider the well-known example of a coin flip. The chance of drawing head or tail carries considerable risk. In the case of a biased coin, however, our views are updated as new observations are collected, which is referred to as the uncertainty. When dealing with feature vectors around the decision boundary in binary classification tasks, this "risk" phrase is appropriate. As a result, MC-dropout generates flipped outputs for highly close classes. Additionally, the test points that occur between two overlapping classes reveal aleatoric uncertainty. As a result, the terms risk and aleatoric uncertainty are somewhat interchangeable. The model risk and aleatoric uncertainty of the predictions are irreducible in both forms. This example will be demonstrated later



(a) Selected Feature: Population; a) Lightweight Regular model b) Heavyweight Regular model c) Lightweight MC model D) Heavyweight MC model



(b) Selected Feature: Median Income; a) Lightweight Regular model b) Heavyweight Regular model c) Lightweight MC model D) Heavyweight MC model



(c) Selected Feature: Total Rooms; a) Lightweight Regular model b) Heavyweight Regular model c) Lightweight MC model D) Heavyweight MC model

Fig. 2: Representation of Uncertainty using Different Models and Features

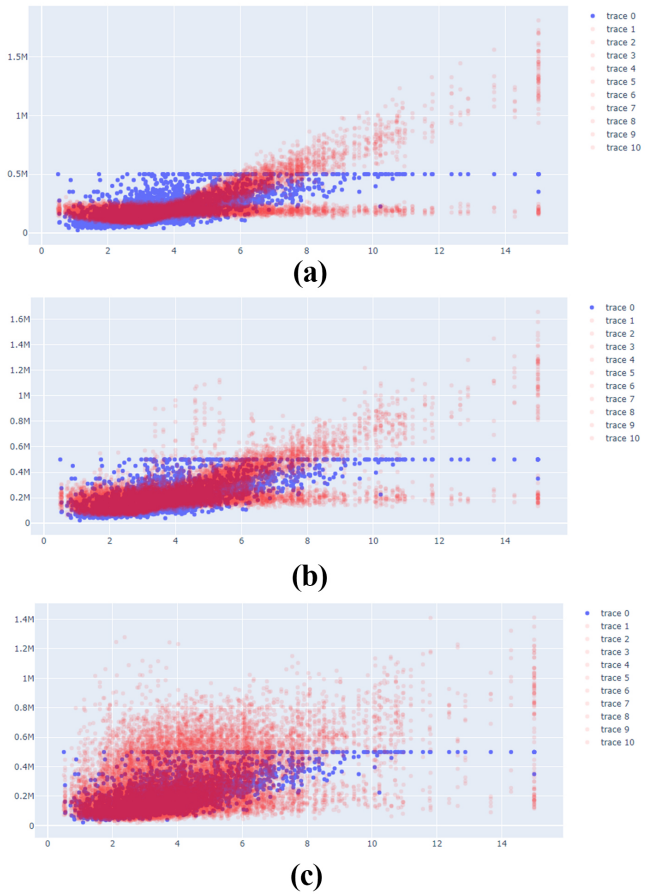


Fig. 3: One Feature (a) vs Multiple Features (b) vs All Features (c)

in this paper's experiment.

Shamsi et al. [26] combine cross entropy with Expected Calibration Error (ECE) and Predictive Entropy to present two novel loss functions (PE). The obtained findings clearly illustrate that using the new suggested loss functions, a calibrated MC-Dropout approach may be obtained. The new hybrid loss functions had a big impact on reducing the overlap between the distributions of uncertainty estimates for accurate and wrong predictions without affecting the model's overall performance. This improves the MC dropout by providing better performance.

6 Conclusion

This technique of representing uncertainty using MC dropout is very simple and very useful. Dropouts are typically used to avoid model overfitting but this technique also allows us to represent the uncertainty of the model. MC sampling is flexible since it can be used to any model type and architecture and is simple to implement. Moreover, MC dropout effects in terms of model repeatability. Monte Carlo dropout also offers a wide range of practical applications. Bayesian neural networks and ensemble-based techniques are two more ways proposed to quantify model uncertainty. The benefit of Monte Carlo dropout over these methods is that no modifications to the model training procedure are required, whereas both of these alternatives result in a significant increase in training complexity. The results we get from this experiment further establishes the importance of Monte Carlo Dropout.

References

1. Gal, Yarin & Ghahramani, Zoubin. (2015). Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. Proceedings of The 33rd International Conference on Machine Learning.
2. Neal RM (2012) Bayesian learning for neural networks, vol 118. Springer, Berlin.
3. Guo, Chuan & Pleiss, Geoff & Sun, Yu & Weinberger, Kilian. (2017). On Calibration of Modern Neural Networks.
4. Kendall, Alex & Gal, Yarin. (2017). What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?.
5. Graves, A. (2011). Practical variational inference for neural networks. In Advances in neural information processing systems, pages 2348–2356.
6. MacKay, David J. C. (1992). Bayesian Interpolation. Neural Computation, 4(3), 415–447. doi:10.1162/neco.1992.4.3.415
7. Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2017). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In International Conference on Learning Representations. arXiv: 1609.04836.
8. Chen, T., Fox, E. B., and Guestrin, C. (2014). Stochastic Gradient Hamiltonian Monte Carlo. In International Conference on Machine Learning. arXiv: 1402.4102.
9. Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin dynamics. In Proceedings of the 28th international conference on machine learning (ICML-11), pages 681–688.
10. Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. In International Conference on Learning Representations.
11. MacKay, David J. C. (1992). A Practical Bayesian Framework for Backpropagation Networks. Neural Computation, 4(3), 448–472. doi:10.1162/neco.1992.4.3.448
12. Maddox, Wesley & Garipov, Timur & Izmailov, Pavel & Vetrov, Dmitry & Wilson, Andrew. (2019). A Simple Baseline for Bayesian Uncertainty in Deep Learning.
13. Mandt, S., Hoffman, M. D., and Blei, D. M. (2017). Stochastic Gradient Descent as Approximate Bayesian Inference. JMLR, 18:1–35.
14. Chen, X., Lee, J. D., Tong, X. T., and Zhang, Y. (2016). Statistical Inference for Model Parameters in Stochastic Gradient Descent. arXiv: 1610.08637.

15. Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In Advances in Neural Information Processing Systems.
16. Kuleshov, V., Fenner, N., and Ermon, S. (2018). Accurate Uncertainties for Deep Learning Using Calibrated Regression. In International Conference on Machine Learning, page 9.
17. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
18. Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras
19. Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... others. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825–2830.
20. Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. Scotts Valley, CA: CreateSpace.
21. Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, University of Cambridge, 2016.
22. L. Zhu and N. Laptev, "Deep and confident prediction for time series at Uber," arXiv preprint arXiv:1709.01907, 2017.
23. Jungo, Alain & McKinley, Richard & Meier, Raphael & Knecht, Urspeter & Vera Ramirez, Luis & Pérez Beteta, Julián & Molina-García, David & Pérez-García, Víctor & Wiest, Roland & Reyes, Mauricio. (2018). Towards Uncertainty-Assisted Brain Tumor Segmentation and Survival Prediction. 10.1007/978-3-319-75238-9_40.
24. Miok, Kristian & Nguyen Doan, Dong & Zaharie, Daniela & Robnik-Sikonja, Marko. (2019). Generating Data using Monte Carlo Dropout.
25. Osband, I. (2016). Risk versus Uncertainty in Deep Learning: Bayes, Bootstrap and the Dangers of Dropout. Workshop on Bayesian Deep Learning, NIPS .
26. Shamsi, Afshar & Asgharnezhad, Hamzeh & Abdar, Moloud & Tajally, AmirReza & Khosravi, Abbas & Nahavandi, Saeid & Leung, Henry. (2021). Improving MC-Dropout Uncertainty Estimates with Calibration Error-based Optimization.
27. Pace, R. Kelley, and Ronald Barry. "Sparse spatial autoregressions." Statistics & Probability Letters 33.3 (1997): 291-297.